Our Ref.: **36-1533**

U.S. PATENT APPLICATION

Inventor(s): Simon G. Thompson
Brian R. Odgers

Invention: TASK MANAGEMENT

NIXON & VANDERHYE P.C.
ATTORNEYS AT LAW
1100 NORTH GLEBE ROAD
8TH FLOOR
ARLINGTON, VIRGINIA 22201-4714
(703) 816-4000
Facsimile (703) 816-4100

SPECIFICATION

TASK MANAGEMENT

The present invention relates to task management and particularly to the ordering and customisation thereof.

Task management is often at least partially automated these days, particularly scheduling and resource allocation. However, the generation of appropriate task sequences, prior to scheduling and resource allocation, in the light of actual constraints and circumstances, remains a challenge.

It is known to apply automation to processes. For instance, the Enterprise Integration Laboratory of the University of Toronto has published material such as the report "Integrated Supply Chain Management Project" by Mark S Fox, Mihai Barbuceanu, Mahmud Gani and Chris Beck which can be seen at http://www.eil.utoronto.ca/iscm-descr.html. This focuses on the management of business processes as they are executed.

Embodiments of the present invention deal with generation and modification of task sequences and content, for potential use with resource management systems.

References below to "ASOPE" are references to embodiments of the present invention. "ASOPE" is an acronym for "Aspect Orientated Process Engineering" and is the name used for the principle underlying embodiments of the present invention.

Inventive aspects of embodiments of the present invention may be identified from the description and include the subject matter set out in the claims.

A process generation system according to an embodiment of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 shows schematically the process generation system drawing on two repositories and a set of rules in generating a program;

Figure 2 shows a set of objects, some of which share functionality;

Figure 3 shows schematically the process generation system drawing on aspect repositories allocated to two interested parties in generating a program;

Figure 4 shows listings for two programs generated by the process generation system, each appropriate to a different respective interested party;

Figure 5 shows a target service problem management matrix mapped against customer types and service types;

Figure 6 shows a generic process plan relevant to the target matrix of Figure 5;

Figures 7 and 8 show intermediate matrices in generating a service problem management matrix designed to implement the target matrix of Figure 5;

Figure 9 shows a generated service problem management matrix using the system;

Figure 10 shows iterative specialisation of a process as it passes through management domains;

Figure 11 shows a use case diagram of the system demonstrating interaction with users;

Figure 12 is a class diagram of an embodiment of the process generation system; and

Figures 13 to 15 show an alternative embodiment of the invention.

Processes can be of different types. A computer process, in particular a computer program, is compiled and run as logical instructions to be translated and acted on by a machine. Other processes are instantiated as physical systems involving people. In particular, there are inputs and outputs to allow decision making, optimisation and customisation by users which will affect the process as performed to produce for instance local, individual and/or time based variations. It is these other processes which ASOPE is particularly adapted to generate and support.

ASOPE decomposes processes into two elements, Generic Process Patterns (GPPs) which are the fundamental sequences of steps which are used to achieve a goal, and Process Aspects which contain context specific process steps together with instructions for their composition with a particular GPP. In this way it is possible to use ASOPE to synthesise wholly new processes that achieve a goal but, with respect to one another, utilise extra or alternative process steps for example to achieve unrelated sub-goals or prevent the system from reaching a particular state. An example of a GPP would be a plan to instal customer premises equipment. Process Aspect examples would be access checks for premises related to an individual, or checking that the site could support physical diversity for a corporate working in the finance industry. GPPs are

decontextualised, generalised plans and Process Aspects provide context, resource and participant dependent exceptions and extensions.

A primary objective of decomposing processes in this way is to construct a generative representation that facilitates the reuse of process elements developed in the past, and "must do" elements derived from local working practice or policy decisions. A secondary objective is to develop a representation that is capable of being structured in a way that reflects the real structure of the organisation which generates and maintains the knowledge.

The use of process aspects which are first class objects within ASOPE enables the development of a process design meta-process that allows information on the current context of the business process, that will affect its structure, to be woven into it at runtime. Also use of a cross cutting abstraction like aspect-orientation has an impact on the taxonomy of process objects that an ASOPE style system would implement.

Object orientation, as used in ASOPE, is the idea that modules in programs should contain instructions and data definitions that relate to one logical abstraction in the domain of the program. Each module should represent the data and operations that concern just one thing or conceptual entity that the programmer, designer or analyst has described. Object orientated modules are called classes, because they define a class of objects.

For example if one were building a program that ran a company pay role using object orientated techniques you would write a class for the company's employees, a class for tax information and some business logic classes representing the way that the system is to make payments. Each employee would be represented within the system by an *object instance* which would be a realisation of the appropriate class, so the tax office would be represented by just one instance of the tax class.

The power of the object orientated abstraction as far as programming is concerned is the hierarchical typing of the classes which enables the programmer to generalise about types of objects in the system. For example if the company wishes to employ a student trainee who is exempt from taxation it would be possible to write a class defining *non-taxable-employees* which would inherit all the properties of a normal employee. However, this class would include different code to be used to calculate the final net pay of the

employee, this time without tax. The advantage of this is that if the normal employee class is changed in other ways, improving holiday allowances for example, then the inheritance relationship between the normal *employee* class and the *non-taxable-employee* class will ensure that the changes will be propagated to it as well. These inheritance relationships can be used to greatly simplify the conceptual structures of the programs thus making them easier to develop, and sometimes easier to maintain.

Although object orientated abstractions have been extremely successful in reducing the whole life cost of software systems, when a larger more complex implementation has been attempted drawbacks with the object orientated abstraction have become apparent. Problems include:

Object schizophrenia. In order to handle the complex behavior of a particular logical entity in a system it is often necessary to decompose the entity into many smaller more easily written classes. The entity is intended to be one class; but, instead, it is comprised of multiple classes, each with its own identity once instantiated into an object. While this tactic allows the designer or programmer to conceptualize the implementation clearly as they construct it, and helps them to manage the detail of the implementation, it makes the task of an outsider attempting to comprehend the system and the function of its parts far harder. This can lead to many symptoms of a bad program, including broken delegation and broken assumptions.

Composition and interactions constraints (and inconsistencies). Object orientated systems impose a number of constraints on the way that objects can be composed and can interact. These constraints are largely arbitrary and are not standard across all object orientated languages and implementations. For example in C++ the concept of a friend function allows a more sophisticated model of interaction between objects to be defined than is possible in Java. Inconsistencies arise because conflicts can arise from different design principles which one wishes to use to construct a program. Videira Lopez (1997) points out in "D: A Language Framework for Distributed Programming", a PhD thesis for the College of Computer Science, Northeastern University, that if one were to make variables of a base class available to a sub-class then the sub-classes can make policy decisions based on the object state; but this violates the referential integrity of the base class and is discouraged in frequently cited design guidelines.

Interface bloat or, alternatively, downcasting. In order to maintain the object orientated abstraction it is often necessary to implement classes that have several different roles for different elements of the system. In the payroll system discussed so far employees would have one relationship to an object with responsibility for calculating a pay bill, and another relationship to an object with responsibility for calculating pension contributions (they might implement a *pensioner* interface for example). These multiple relationships and roles force the programmer to implement several interfaces, or sets of function calls in significant objects of the system which increases their size, and muddies their function. Downcasting is an alternative strategy which enables a programmer to handle this kind of problem by instructing the program compiler to regard an object as being of different types at different points in the programs thread of control. However downcasting is a dangerous practice for two reasons. Firstly the semantics of a down cast are different between various object orientated languages (for example in C++ a downcast leads the compiler to execute the method of the class that the object has been downcast to, but in Java the method will be the actual method implemented for the objects "real" class). Secondly, downcasts violate the type checking support that is one of the major benefits of object orientated technology.

Cross Cutting. Behaviours that are required throughout an entire system, such as particular safety protocols or quality of service assurance procedures, often need to be re-implemented in multiple places if they are to be integrated with a object orientated program. This kind of required behavior is said to *cross cut* the objects. Cross cutting leads to tangling (Kiczals et al June 1997, in "Aspect-Oriented Programming", proceedings of the European Conference on Object-Oriented Programming published by Springer-Verlag). A tangled program is one in which concerns that should be encapsulated together are scattered across a number of modules. Tangled code is difficult to locate and change during maintenance.

These problems have encouraged software engineers to develop other abstractions that are better fitted to the kind of encapsulations that are required for the development and maintenance of large and complicated computer systems in environments where resource constraints and time to market pressures are paramount.

Subject orientated programming" (by Osher & Harrison 1992, published in "Combination of Inheritance Hierarchies", Proceedings of 1992 Conference on Object-Orientated

Programming Systems, Languages, and Applications) addresses the fact that different entities view the same object from different perspectives. These perspectives are not filtered views; some properties and behavior only exist because of the perspective. Subject oriented programming (SOP) provides extra language features above and beyond object oriented programming to support subject decomposition, implementation, and re-composition. In SOP, a subject is defined as a collection of classes or class fragments that models its domain or area in its own, subjective way.

Although subject oriented programming appears to be a considerable advance on object orientated programming, there is another approach, aspect oriented programming, that provides a more generic method of integrating cross cutting behavior than SOP.

Aspect oriented programming also adds extra language features to object oriented programming. Aspects cut across or *cross cut* the units of a system's functional decomposition (objects). Examples provided in the literature are synchronization, exception handling, monitoring and auditing, quality of service, and many others.

Referring to Figure 1, research has produced language constructs and compilers (called Aspect Weavers 100) that can interleave or *weave* component definitions 105 (objects) and aspect definitions 110 (programs) appropriately to formulate a unified and executable program 115.

To implement the embodiment of the present invention described below, the Java based AOP environment (AspectJ ™) from Xerox PARC and a known AOP language are used. Information on these is available for instance published by Videira Lopes, C. & Kiczales, G in "Recent Developments in AspectJ™", in ECObject OrientatedP'98 Workshop Reader, Springer-Verlag LNCS 1543.

Referring to Figure 2, the genesis of aspects may be more clearly understood if one considers five objects, Obj1 - Obj5, involved in three activities , [A],[B], and [C]. If the objects are instances of five different classes, the functionality required to carry out the activities *cross cuts* method definitions (potentially many) in five separate classes.

Aspects are first class objects, which are woven into a program at compile time, but with their own independent state. As is implied by their status as first class objects, an aspect

can be woven with several different objects within a program and the state of the aspect is shared and updated by all of those objects that are woven with it.

Processes are sometimes required to be flexible and dynamic. However some processes remain cumbersome and difficult both to construct and comprehend. This is because of the numerous "special cases", exceptions and decision points that must be included in a realistic process implemented by a large organisation that sets the needs of its customers as a priority. For example a process developed to provide International Private Circuits for customers of BT (the applicant) contains sixteen decision points on the top level process; sixteen top level process tasks and each of the top level processes contain numerous decision points and exceptions themselves. Research techniques, such as agents in business processes, solve numerous problems in the provisioning and enactment stages of processes but the flexible creation of processes, given the needs of the participants (consumer, employee, regulator, etc), has not been developed in the same way.

ASOPE considers the construction of specialised processes from generic cores and context dependent elements. New process steps or different process sequences can be implemented as new policies and instructions are disseminated across a domain. The essence of the process is abstracted from the detail of customer preferences, local working agreements and management instructions and then reused for different customers, workplaces and management domains.

Referring to Figure 3, in order to abstract the core of the process from the context or customer dependent elements that must be considered, the process is defined in terms of two sets of components:

Generic Process Patterns 300: partially ordered processes, which consist of process steps, which are obligatory for a particular goal to be achieved. These are an ordered list of the conditions that must be satisfied for the goal to be successful; and

Process Aspects 305: process steps that can be included in a business process in order to customise it for execution by a particular resource.

A generic process pattern 300, a core process that must be executed in order to achieve a type of goal, comprises a set of processes that has been loaded to the system for instance because they are publicly advocated for use by management and subscribed to by operational units.

The process aspects 305 are then context dependent process steps that should be carried out in order to achieve a particular goal for instance allocated to a particular team, in a particular place for a particular customer. Process participants are provided with interfaces 310 in order that they can load their particular process aspects 305 to be woven by the aspect weaver 320 into the generic process 300 to create a specialised ordered process 315 for execution.

The process aspects 305 used are a subset of an overall available set of aspects and the process context determines their use. The aspect weaver 320 does not just append process steps onto another process. It merges aspects 305 with the GPP 300 according to weaving rules instantiated by the aspect weaver 320 and weave instructions that are coded into the aspects 305 themselves.

Aspects 305 can *introduce* new process steps, and they can *advise* existing process steps. An *advise* weave appends new process instructions into a particular place in the process. The process position that an aspect's advise weave is to be woven at is determined by the weaver 320 matching patterns defined in the aspect 305 to a process step name in the GPP 300 with wild cards matching to the preconditions and post conditions of the process step.

The processes are arranged hierarchically using a frame based object orientated organisation, allowing this kind of match and weave process to be used to merge process sequences together.

In the following are described two examples of how this can be applied to process activities.

Example 1: Customised Processes for Telephone Installation

Many commercial organisations have a core activity that concerns the installation or servicing of equipment at a customer's site or domicile. For example: gas, water and electricity utilities must install, service and read meters in order to bill their customers.

Consider the example of installing a phone, a business process that has some significance for BT. A simple set of core generic process steps are required for all

customers, no matter where they are in the country, or which engineer does the installation. However, some steps will be different from customer to customer. A corporate customer may require adherence to a previously agreed contract, breach of which may be unacceptable. Providing a service in a rural area may involve more travel time, but it will be unnecessary to book a parking bay for the engineer's van, however in the City of London parking space may need to be pre-arranged.

In order to apply ASOPE to the process of installing a telephone it is necessary to define the participants and the generic process of installation (the GPP 300) to be used. For the purposes of this example two different (simplified) customer types are considered.

Customer type 1: Customer's time is paramount.

Customer type 2: Customer's security is paramount.

While the goal of installing a telephone is identical for both of these customer types, they are qualitatively very different. Customer type 1 will place a premium on appointments being kept and probably can supply a mobile phone number that allows them to be contacted a short time before the technician arrives to carry out the work. Customer type 2 will be available for the visit of the technician and will require a special identification procedure to reassure them that the technician is genuine.

A single process can be used to achieve the goal of installation for both groups of customer. However, by localising knowledge of extra process steps each customer requires in a process aspect 305, the process can be untangled to provide a customised process for each of the customer types.

An aspect 305 for each customer type to be considered is defined. An aspect hierarchy could be created that can be used to inherit behaviour, but for this example the two customer aspects are considered in isolation.

```
aspect CustomerType1 extends Customer{
int customerPhoneNumber = 0;
 advise
 *installPhone.bookVisit(*) {
  after {
  System.out.println("Acquire customers
  mobile contact number");
  customerPhoneNumber = 12345678;
        }}
 advise
   *installPhone.installEquipment(*) {
before {   System.out.println("Verify
appointment with customer number = " +
String.valueOf
(thisAspect.customerPhoneNumber)); }}}


aspect CustomerType2 extends Customer{
 advise *installPhone.installEquipment
 (*){
   before {System.out.println("Verify
     ode word with customer"); }}
 advise * installPhone.bookVisit(*){
   after {System.out.println("Supply
     code word to customer"); }}
```

```
public void acquireJobDetails() {
  System.out.println("Acquire
  appropriate  job information");}
public void bookVisit() {
  System.out.println("Book visit time
  and address, allocate engineer"); ; }
public void waitUntilVisit() {
  System.out.println("Wait for visit
  time"); }
public void installEquipment() {
  System.out.println("Carry out
  appropriate installation work");}
public void billCustomer() {
  System.out.println("Send bill to
  customer"); }


public void planPattern () {
        acquireJobDetails();
        bookVisit();
        waitUntilVisit();
        installEquipment();
        billCustomer(); }
```

In the above, the left hand side shows instantiated aspects in relation to CustomerType1 and CustomerType2 and the right hand side shows a generic process pattern for phone installation to be used as the basis for the process. The aspects are instantiated, as is the process pattern, in an Aspect-J™ module. Each customer's process aspect is then added to the process pattern in turn to produce two customised processes, shown in Figure 4.

It can be seen that the aspect in relation to CustomerType1 contains sufficient information for the aspect weaver 320 to locate where and how to modify the GPP 300 for phone installation shown on the right hand side. In particular, it tells the aspect weaver 320 to extend the GPP 300 by adding a print line "Acquire customers mobile contact number" at the end of "bookVisit", to add the print line "Verify appointment with customer number....." before "installEquipment", and it instantiates the customer's fixed network telephone number 12345678.

With respect to the aspect in relation to CustomerType2, this one tells the aspect weaver 320 to extend the GPP 300 by adding the print line "Verify code word with customer" before "installEquipment".

Even the simple examples given above show the need for aspect instances. Customer Type 2 requires that a code word is set for each appointment. It is appropriate to associate this information with the customer, and an aspect instance is an appropriate way of handling this.

The following section describes another example of how a process can be decomposed into generic process plans and process aspects. This example describes the problem of structuring a service provision process that is dependent on both the type of customer buying the service and the type of service being sold.

Providing and managing communications services is a key activity for communications providers such as the applicant. A large portfolio of services of various types can exist, with each customer requiring a different subset of provision in order to conduct their business. In order to provide a service it is necessary to provide methods that are appropriate for its management and maintenance at a level acceptable to the customer. Unfortunately customer requirements and resources vary considerably. For instance, it is inappropriate to provide a "five nines" (99.999%) availability of a service at the cost of thousands of pounds each year for an individual. The individual can accept a much higher level of unavailability and cannot afford the charge for the service with this level of maintenance. On the other hand, a corporate customer may view thousands of pounds a year as a small price to pay for high availability of a business critical service such as voice mail. Also, it may be against the service provider's interest to provide a lower level of availability for a business critical service because of the impression of unreliability that it would create with a valuable corporate customer.

Figure 5 shows a matrix of various problem management strategies that might need to be applied to service types for differing customers. The matrix contains twelve elements representing the requirements for problem management given a particular type of service for a particular customer. Each of the entries in the matrix indicates a type of problem management that is applicable to the context mapped by the service type and the customer type. There are six different problem management strategies that might be adopted.

Replace indicates that a service provider, acting as a broker between the network company providing the network infrastructure for the service and the customer, will replace the service rather than attempt to discover the problem and fix it.

C-Enabled is a customer enabled problem management strategy, where the customer is provided with a suite of diagnostic and trouble shooting tools which enable them to manage problems as they arise in the service.

Servicing is for the service provider to attempt to avoid faults in the service to prevent its failure, in much the same way that a car would be serviced to prevent a breakdown.

PM-enabled (not shown) posits the existence of self diagnostic abilities for the service that allow it to predict and report impending failures that will allow preventative action to be taken (the PM in the name is taken from Photocopier Model, a reference to the ability of modern office automation systems to phone faults through to their suppliers).

Open Testing provides the customer with the ability to test for faults on the service, but requires that the network connector provides the problem solving activity.

3rd party relies on the existence of a specialised 3rd party rescue service, similar to breakdown and recovery organisations that motorists join and use if their cars fail.

The goal of the GPP here is to be able to prepare a quote for providing a particular type of service to a particular type of customer taking into account the amount of problem management that it is appropriate for the customer to do themselves. It is possible to achieve the goal by using an aspect orientated representation to fold the matrix into its axis and represent the problem management strategies as process aspects representing the *type of service type* and the *type of customer* for that service. These can then be woven together, with a generic process pattern representing the process (shown diagrammatically in Figure 6) to generate service provision processes tailored to the customer, and to the service that is to be provided.

In order to be able to do this, the elements need to be defined that will be used as knowledge stores. Referring to Figure 5, the aspects can be identified that will be required to store the knowledge to customise the business process. There are three types of customer 500 (Corporate, SME and Individual) and four types of service 505 (niche-dispose, niche-value, mass-dispose, mass-value). All of the process aspects 500,505 (the service types and the customer types) contain *advise* weaves (equivalent to that shown above for the telephone installation example) which introduce extra process information to the generic process plan of Figure 6 at the "produce estimate" step 600.

Referring to Figure 7, by using just the customer type aspects 500 to contextualize the process, a matrix is produced (selecting the most repeated matrix elements for each of the columns).

Alternatively, referring to Figure 8, by using just the service type aspects 505, a different matrix can be produced. The aspect information here was generated by selecting the most commonly shown strategies for problem management in the matrix co-ordinates as shown in Figure 5, selected in order to generate a best fit to the original matrix. However, the niche-value and mass dispose categories both contained a different strategy for each customer type. This forced an arbitrary choice for one of the strategies. It is preferable to use domain knowledge to determine which strategy would be best if applied for all customers.

Both Figures 7 and 8 show that weaving across one axis of the problem management matrix can be used to specialise a generic process to some degree. However, in order to regenerate the original matrix shown in Figure 5 both sets of aspects need to be woven into the plan. In order to do this, a set of precedence relationships is needed which can be defined against the aspects, which will force the weaver to make the correct decision as to which strategy to choose. Alternatively if the aspect weaver operated according to some known weave ordering rules the same effect could be achieved by determining the correct order of weave for the various aspect categories. Effectively an ordered weave enables implementation of the matrix shown in Figure 9.

This weave can be achieved with the following algorithm:

- Using service-type-aspects introduce process steps for *estimatingReplaceCost* for the niche-dispose type, *estimateOpenTestingCost* for the nicheValue service type, *estimateReplaceCost* for the mass-dispose service type and *estimateServicingCost* for the mass-value type.
- Use advise weaves to place calls to these process steps into the generic process pattern step
- Introduce a process step for *estimateServicingCost* for the corporate customer type.

- Use an advise weave to the *estimateReplaceCost* process step (introduced by the previous weave) to add the *estimateServicingCost* step to the process[1].
- Introduce a process step for *estimateCustomerEnabledCost* in the corporate customer type.
- Use an advise weave to the *estimateOpenTesting* step introduced for the nicheValue service type to force this step into the specialised process.
- Introduce a process step for *estimateCustomerEnabledCost* in the individual customer type.
- Use an advise weave to the *estimateReplaceCost* to force this step into the process.

This algorithm is sufficient to generate the matrix of problem management strategies appropriate for the particular customer using the particular service as shown in Figure 5 except that the corporate-mass-dispose co-ordinate of the matrix is wrongly populated with a servicing policy. This is because there is a conflict between the exceptional case of replacing an *estimateReplaceCost* step for a corporate with a *estimateServicingCost* and the exceptional case of replacing an *estimateReplaceCost* step for a corporate with a *estimate3rdPartyCost.* The price of using this method is the incorrect entry in the matrix, but the benefit is that it has been possible to generate the twelve elements of the matrix from seven process aspects.

In the following section, the architecture of a process knowledge management system that would utilise ASOPE as its central component is described. The architecture is examined from two perspectives. The perspective of the organisational hierarchy of the system shows the logical structure of a Process Aspect repository. The business role perspective shows the way in which the system would interact with the participants in the business process in order to be of use.

Organisational Hierarchy Perspective

The assumption underlying this perspective on the architecture is that the participants in the meta process of constructing and managing the enactment of the business process belong to particular management domains which are controlled by managers with limited, but real and valuable, autonomy. This enables the participants to determine which resources will be utilised to achieve a goal on the basis of the impact that their use will

[1] It should be noted that using the Aspect-J weaver it is necessary to also add a return() statement to force the process step to finish execution at this point.

have on the characteristics of the process. Figure 10 shows the process elements specialised by an ordinate layer 1000 being passed to a sub-ordinate layer 1005 to incorporate its process knowledge.

During each weave a process aspect can customise the process in two ways:

Process addition: A process step is added either before or after a previously defined step.

Process redefinition: A process step is redefined to make it relevant to the business process context.

User Role Perspective

Figure 11 shows a model of how an ASOPE system would be used within a commercial organisation. The model is shown as a UML use case diagram in which users of the process are shown as stick figures 1100 and use-cases (in this case activities in the process construction meta-process) are shown as ovals 1105. Notes 1110 are given on the interaction connections between the use cases 1105 and the users 1100. An operational support system 1115 is shown as a "class".

Figure 12 shows a class diagram of components for building an embodiment of the present invention.

Referring to both Figures 11 and 12, the user interacts with the system, particularly to enter information, in three ways:

1) As a Goal Setter 1120 by setting a goal on the system which results in the automatic selection of a GPP to achieve the goal and the selection of resources to provision the GPP.

2) As a Process Writer 1100 by describing processes to the system in terms of GPPs and Aspects, via the Process Editing GUI 1215

3) As a policy maker 1125 describing policies that are integrated with the aspects via the policy editing GUI 1225.

The data stores (not shown) for holding entered information are the GPP library and an aspect library. The GPP library contains a set of descriptions of how a business goal can

be achieved in terms of the sequence of activities that must be undertaken regardless of the context of the goal or the resources uses to achieve it. The "index" for GPPs would be the goals that they achieve. The Aspect Library is where process aspects will be stored. These aspects are the context and resource specific activities and are indexed on the resource or context that utilizes them.

Figure 12 shows a class diagram of the components that might be deployed in implementing an embodiment of the present invention. An important point is that processes for construction are stored in two orthogonal hierarchical repositories, the aspect manager 1200 and the GPP manager 1205. Information is drawn from these and recombined to create processes that achieve goals specified via a goal setting interface 1210, tailored according to actual resource limitations monitored by a resource manager 1235.

The important point about Figure 12 is that the processes are stored in two orthogonal hierarchical repositories (the Aspect Manager 1200 and the GPP manager 1205) from which information is drawn and recombined to create processes to achieve the desired goals. The advantages of this are:

1) Abstraction and representation of the process as a GPP which can be used as a template in multiple contexts.
2) The weaving process can be used to tailor business processes to meet individual goals.
3) The process aspects can be used to implement process steps that meet process constraints that are separate from the business goal, and are the concern of the local management.

The ASOPE approach is dynamic and adaptive in that the cycle of the process knowledge through the system results in the iterative updating and maintenance of the knowledge base.

It should be noted that, in the above, an implementation trick is used (the insertion of a return() statement in the generated program specifying the process) to produce a substitution effect in the weaving. It is believed that a substitution, or filtering weave, is necessary in order for the Aspect Orientated abstraction to be really useful both for

process representation and software development. Composition filters have been used to develop Aspect Orientated systems that directly implement deletion mechanisms which are not implemented in Aspect-J ™ (see M. Aksit and B. Tekinerdogan, Solving the Modeling Problems of Object-Oriented Languages by Composing Multiple Aspects Using Composition Filters, AOP'98 workshop position paper, 1998) and there is no obstacle to the implementation of more weave instructions.

Another characteristic of systems such as that described is the conceptual difficulty involved in the design of the weave orderings, and the correct decomposition and encapsulation of the process elements into generic process patterns and process aspects. This applies to any object orientated or frame based representation, because the essential answer to the question of "why is that the correct object model" is simply "we don't know, but it is". There are no hard rules for the development of object models, simply best practices and rules of thumb which apply only to the particular circumstances of the program and programmer who are structuring a particular model. Aspect Orientation however offers another tool which can be used to simplify the organisation of knowledge.

Variations envisaged in embodiments of the present invention include the provision of graphical support and the development of tools to build, maintain and use process repositories.

In a significantly more flexible variation of embodiments of the present invention, the operation of the aspect weaver 320 can be significantly improved by using an open instruction/action set rather than one which is installed at compile time for the aspect weaver.

Referring to Figures 13 and 14, this shows an iterative process for composing processes using aspect weaving. A set of Aspects 305 is combined with some user defined structure 300. The Aspect 305 contains a tag 1300 which is the instruction to be used, such as "before" or "after", the subject 1305 of the tag, which is a reference to an element in the "general structure to be specialised", and some data 1310 which will be interpreted by the composition algorithm 1400 of the weaver 320.

The composition algorithm 1400 of the weaver 320 takes the information that is parsed and processed by the weaver from the aspects 305 and structures 300 to be specialised

by it, identifies action steps 1310 which aspects 305 determine will be woven into a structure 300 and matches these to actions in its fixed instruction/action set 1405. It then applies the actions from the fixed instruction/action set 1405 that are matched by the appropriate instruction from the aspect 305. In this way the structure 300 that is being processed becomes specialised as shown in Figure 13. This method has the advantage of being simple to use, however, it is also limited to the semantics of the instruction/action set 1405 that is included in the aspect weaver 320. In some implementations the instruction/action set 1405 may not be separated from the composition algorithm 1400, however, in logical terms this makes no difference to the algorithm.

Figure 15 shows an aspect weaver 320 which uses an open instruction/action set 1505, ie not written into the aspect weaver 320. This has two advantages. Firstly the behaviour of the weaver 320 can be specialised to be dependent on the domain. For example, the meaning of the tag "before" 1300 may be different in a real time domain and a data-processing domain. In the real time domain, it is essential that the elements in the aspect 305 that is composed are executed before the start time for the elements being specialised. This is a tight constraint. In the data processing domain the start time for the execution may be irrelevant, and all that is important is the ordering of execution.

A second advantage is that instructions that have no meaning for some domains may be introduced into the weaver 320. For example, in a computer system that has no multi-threading a weave instruction 1310 "in-parallel" is nonsensical. However in a multi-threading computer system it has a clear meaning (that a thread should be started to execute the elements in the aspect, and that the current thread should be suspended until that thread can be started). In a process weaving system it has another clear meaning that the start conditions for two parallel processes should be made true.

The rules in the instruction/action set take the form of [t,*sub'= f(m,sub)*] where *t* is the tag 1300, and *sub'= f(m,sub)* is the function to apply to the subject data in this case. The weaver can then apply the instruction/action set as in Figure 13. At each step the subject *sub* is transformed into *sub'*, and *sub'* is substituted for *sub* to specialise the structure. This is repeated until there are no matching conditions 1305 from the aspects 305 which can be applied to any part of the structure 300 being specialised.

The function *sub'= f(m,sub)* from the open instruction/action set therefore determines the meaning 1310 of the tag *t* at weave time. It is executed using standard reflection procedures (for example the mechanisms in java.lang.reflect.Method, java.lang.Class & java.lang.Object[2]) by the composition weaver. This contrasts to the scheme in Figure 14 in which the meaning of *t* is determined at the compile time of the weaver itself.

Identification of the domain, and therefore the correct *sub'*, could be done in one or two ways, for instance by a data input by a user at runtime or via the index used for a generic process plan in storage.

[2] See Flanagan,D, 1999, "Java in a Nutshell", O'Reilly & Associates, Inc. ISBN : 1-56592-487-8